TESTOVANIE A JEHO RADOSTI

Testovanie je ako výchova, ak sa zanedbá výsledok si odnesieme v budúcnosti.

Roman Pipík

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
pipik.roman@gmail.com

Abstrakt. *V eseji sa zaoberáme výhodami a nevýhodami softvérového testovania, pričom sledujeme čo komplikuje a čo zjednodušuje testovanie v súčasnosti. Nachádzame aj niekoľko riešení základných problémov testovania. Sledujeme možnosti testovania v malom tíme pracujúcom na krátkodobom projekte. Snažíme sa nájsť metódy testovania ktoré uľahčia a zefektívnia testovanie v tomto tíme. Opisujeme metodológiu vývoja projektu SCRUM, ktorá abstraktne definuje prístupy k testovaniu. Zhodnotíme tiež výhody a nevýhody testami riadeného vývoja, ktorý táto metodológia často využíva. V závere sa snažíme navrhnúť alternatívny prístup k testovaniu, ktorý nie je v súčasnosti ešte veľmi zaužívaný, Touto alternatívou je testovanie aspektovo orientovaným prístupom, preto sa pozrieme na jeho pozitívne vlastnosti aplikovateľné na testovanie.*

Kľúčové slová: *kvalita, testovanie, SCRUM, jednotkové testy, testom riadený vývoj, efektívne testovanie, aspekty, aspektovo orientovaný vývoj*

Význam a úloha testovania

Oproti počiatkom softvérového testovania sa v dnešnej dobe testovanie technologicky podstatne zjednodušilo. Komplexnosť súčasných projektov je však oveľa zložitejšia a preto testovanie zostáva stále rovnako náročnou úlohou ako na začiatku ak nie ťažšou. Najprv si vysvetlime čo to testovanie vlastne je.

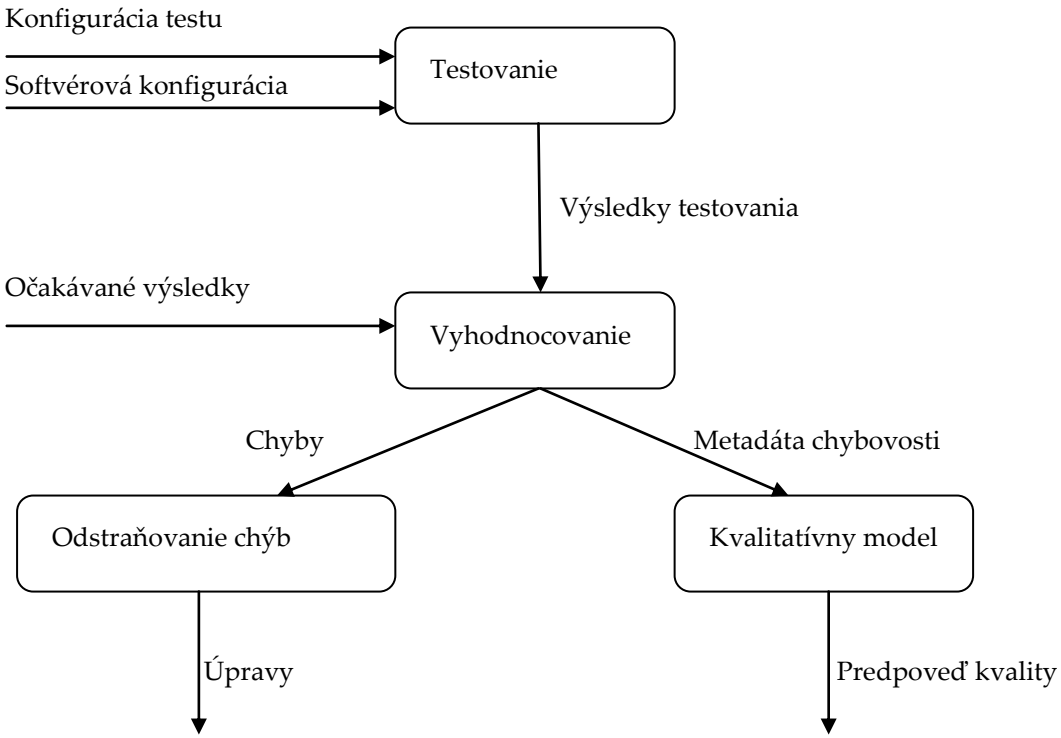
V roku 1950 napísal „Alan Mathison Turing „článok ktorý sa označuje za prvý článok venovaný testovaniu. Z tohto článku vyplýva otázka „*Ak máme vytvoriť konkrétny*

program, ako vieme že spĺňa dané požiadavky?". Či program spĺňa požiadavky sa dozvedáme práve testovaním. Miller definuje testovanie takto [1]: "*Cieľom testovania je potvrdiť kvalitu softvérového systému systematickým overovaním softvéru v dobre kontrolovateľných podmienkach*". Testovanie je tu rozumne ohraničené na kontrolovateľné podmienky. Toto ohraničenie je spôsobené nedokonalosťou testovania, nakoľko testovanie nikdy nebude schopné pokryť celý softvérový systém v respektíve všetky možné stavy systému. Dôvodom je abstraktnosť softvérového systému a nepredvídateľnosť podmienok za ktorých sa bude systém používať.

Úlohou testovania je odhaliť chyby – odchýlky od očakávaného správania pri kontrolovaných podmienkach. Predstavme si prípad testovania používateľského rozhrania, kde test odhalí posunutie ovládacieho prvku na obrazovke o niekoľko bodov. Ak užívateľ túto chybu nepozoruje, nakoľko ľudské zmysly nie sú dokonalé, tento test by sa dal považovať za zbytočné mrhanie časom vývojára. Avšak v niektorých prípadoch by táto chyba mohla mať aj vážne následky ako prekrytie iného ovládacieho prvku a podobne. Preto by som úlohu testovania definoval trocha zložitejšie a to nasledovne: „*Úlohou testovania je odhaliť všetky pre systém významné chyby, a kategorizovať ich podľa dôležitosti a podstaty*".

Komplikácie a zjednodušenia testovania

Aby sme testovanie definovali úplne načrtneme si proces testovania [5] na Obr. 1. Schéma procesu testovania. Na obrázku je možné vidieť všetky vstupy, stavy a výstupy testovania. Je možné sledovať komplexnosť procesu testovania, čo je častou nevýhodou a dehonestuje funkciu testovania ako náročnú a nechcenú činnosť. V procese testovania môže nastať viacero chýb, od chyby v testovacom skripte, nesprávnych požiadaviek na test až po nepochopenie výsledku testovania. Preto by za testovanie ako celok mali zodpovedať skúsení softvéroví návrhári. Na najnižšej úrovni však za testovanie zodpovedá každý vývojár samostatne a to tak že overuje správnosť implementácie ktorú vytvára.



Obr. 1. Schéma procesu testovania.

Z tohto pohľadu vidieť dôležitosť ľudí – vývojárov v procese testovania, a vynárajú sa nám dva problémy. Prvým problémom je samotná nedokonalosť ľudí a nech je prístup akýkoľvek, vždy budú v tomto procese subjektívne chyby. Riešením je automatizácia procesu testovania a prenesenie veľkej miery zodpovednosti na vývojové prostriedky. Druhý problém je subjektívny pohľad vývojára na testovanie, čo môže spôsobiť nepochopenie spôsobu testovania, a s tým spôsobený odpor k testovaniu. Príklad môže vyzerať nasledovne: Vývojári pracujú v páre, a navzájom si prehliadajú program, čo môže niektorý z nich interpretovať ako „*Ten druhý musí po mne opravovať chyby!*“. Riešenie tohto problému už nespočíva len v použití vývojových prostriedkov, riešením je ozrejmenie testovania a všetkých jeho náležitostí predtým ako sa začne projekt vyvíjať. Súčasťou tohto ozrejmovania by malo byť aj určenie prostriedkov použitých na testovanie.

Počas projektu môže dôjsť aj k úplnej neochote testovať. Myers označuje hlavné príčiny tejto neochoty ako [2]:

- Ego programátora – chyby značia že programátor sa mýli
- Vyčerpanie z testovania – testovanie často prebieha pod časovým tlakom a hľadanie chyby môže byť pre programátora psychicky náročné
- Náročnosť opravy chyby – po odhalení chyby musí programátor pátrať v programe kde chyba vznikla, čo môže predstavovať prehliadanie veľkého množstva zdrojového kódu

Testovanie v malom tíme

Výhodou malého tímu je jeho flexibilita, teda je možné aplikovať takmer akúkoľvek techniku testovania. Zatiaľ čo u veľkých tímov je forma testovania často dopredu daná, vyplýva z požiadaviek projektu alebo je určená už zaužívanými technikami, pri malom tíme je možné ľahko zmeniť formu testovania podľa potreby. K tejto zmene musí dôjsť pred začatím vývoja projektu, kedy sa zjednotia rozdielne názory členov na testovanie. Výber formy testovania môže byť prepletený s metodológiou vývoja, preto je vhodné tieto rozhodnutia spojiť. S výberom formy testovania súvisia aj použité vývojové prostriedky, s ktorými sa niektorí členovia musia oboznámiť. Učenie týchto prostriedkov počas behu môže brzdiť vývoj, a je skôr vhodné pre dobre zabehnutý tím, ktorý je ochotný a schopný experimentovať. Začiatok projektu v malom tíme preto môže vyzerať značne edukatívne.

Zatiaľ čo pri väčších tímoch je príčinou neochoty testovať náročnosť opravy chyby či psychické vyčerpanie, v malom tíme je hlavnou príčinou „ego programátora“, kedy programátor nechce byť označený za neschopného, či pomalého člena tímu. Na druhej strane sa programátor snaží o samostatné testovanie svojich častí programu, tak aby jeho výsledok práce bol kvalitný. Je to spôsobné zodpovednosťou jednotlivca za časť programu, pričom pri veľkých tímoch vstupuje do hry viac „kolektívna vina“ za chybu, prepletenie kódu medzi vývojármi a celkovo komplexnosť vývoja v tíme. V ďalšom texte sa pozrieme na viacero aspektov testovania v tomto tíme.

Testovanie a SCRUM

Metodológia SCRUM ako jedna z populárnych agilných metód vývoja považuje testovanie za najdôležitejšiu časť vývoja. Môže zato prístup nazvaný „What is Done“ čiže „čo je hotové“[6]. Vývoj touto metódou si vyžaduje dopredu definovať kedy je časť programu hotová. Ako z úvodu vieme overenie sa vykonáva testovaním

Pred samotným vývojom je potrebné definovať aké požiadavky musí spĺňať úloha nato aby mohla byť splnená na sto percent. Ak úloha nie je stopercentná nie je možné prejsť na ďalšiu úlohu. Vďaka tomu je počet aktuálnych testov ohraničený na niekoľko aktuálnych úloh, a tieto testy sa vytvárajú spolu s aktuálnymi úlohami. Pri iných metódach vývoja zostávajú úlohy čiastočne nedokončené čo zvyšuje potrebu testovania. Ak dôjde k odhaleniu chyby v už ukončenej úlohe, opravenie chyby sa zaradí medzi ostatné úlohy a rieši sa podľa priority. Tento jav výborne reprezentuje rozšírenú definíciu testovania ktorú sme definovali v úvode tejto publikácie. Dokonca môže nastať jav kedy nedôjde k napraveniu chyby nakoľko jej priorita bude nedostatočná pre zaradenie medzi aktuálne úlohy. Pri iných prístupoch by došlo k okamžitému riešeniu chyby bez ohľadu na prioritu. Podľa výskumu [3] je forma testovania používaná pri metóde SCRUM oveľa sympatickejšia a vývojári vnímajú testovanie oveľa pozitívnejšie, a to hlavne z týchto dôvodov:

- nie je potrebné opakovať staré testy
- testy sa vytvárajú tak ako prichádzajú úlohy

Vidím jediný nedostatok metódy SCRUM v oblasti testovania. Predstavme si už ukončenú úlohu, ktorá obsahuje skrytú chybu. Ak sa táto chyba nájde, je potrebné znova preštudovať celú úlohu, nakoľko vývojári už dávno pracujú na niečom inom.

Testami riadený vývoj

Ako sme spomenuli pri opise metódy SCRUM, testovanie je spolu s návrhom a implementáciou najdôležitejšou a základnou časťou počas vývoja. Preto je rozumné zamyslieť sa aj nad vývojom riadeným testami. Jedná sa o techniku využívajúcu jednotkové testy. Vývoj sa vykonáva v iteráciách ktoré sa [4] skladajú z nasledujúcich krokov:

- čiastočný vysoko/nízko úrovňový dizajn
- napísanie niekoľko automatizovaných jednotkových testov
- spustenie testov a overenie že všetky zlyhali (nakoľko žiaden program ešte neexistuje)
- implementovanie programu tak aby boli testy prebehli v poriadku
- znova spustenie testov, aby sme overili že skončia úspešne pre nový program
- reštrukturalizácia programu a testov podľa potreby, tak aby sa program vykonával lepšie či efektívnejšie

Výhody tohto prístupu sú [4]:

- výsledky testu naznačia keď sme nechcene narušili existujúcu funkcionalitu programu
- vývojár môže pridávať funkcie a meniť štruktúru programu bez toho aby sa obával narušenia existujúcej funkcionality
- automatické testy určujú čo je potrebné aktuálne vyvíjať

Mnoho ľudí si testami riadený vývoj zamieňa s prístupom „*Najprv sprav testy na všetko, a potom programuj*“! Tento mylný názor sa spája hlavne s „V“ modelom vývoja, kde sa dlhšia časť úvodu projektu venuje analýze. V skutočnosti sa testy vytvárajú na nízkej úrovni, a mali by sa vytvárať pred implementovaním konkrétnej funkcionality. Takto nie je potrebná komplexná analýza, stačí iba analýza ktorá je potrebná pre implementovanie konkrétnej funkcionality. Pri vývoji metódou SCRUM sa môže jednať o vytváranie testov pre aktuálnu úlohu, prípadne vytvorenie niekoľkých testov pre šprint.

Chcem zdôrazniť že analýza pred vytváraním testov je významným prvkom v úspešnosti tohto prístupu. Bez dostatočnej analýzy je potrebné testy výrazne meniť počas vývoja programu a to predražuje túto formu vývoja. Vývojári navyše strácajú záujem o túto formu vývoja pokiaľ sa viac musia venovať úpravám testovania než úpravám funkcionality programu. Pokiaľ je implementovaná úloha príliš abstraktná je potrebné sa zamyslieť nad jej rozložením na menšie časti. Ak nie je možné úlohu rozložiť, je potrebné zvoliť iný prístup k testovaniu nakoľko pri testami riadenom vývoji môže dôjsť ku skomplikovaniu situácie častými úpravami testovania.

Automatizácia testovania

Pri testami riadenom vývoji sa ukazuje potreba značnej automatizácie testovania, nakoľko vykonávanie testov je časté, a neustále opakovanie testov môže byť pre vývojára veľmi odradzujúca úloha.

Úlohou vývojových prostredí pri automatizácii testovania má byť:

- zjednodušiť proces testovania, najlepšie na jednoduché kliknutie tlačidla testovať
- zachytiť stav procesu pri odhalení chyby
- zobraziť chybu a kategorizovať ju
- vytvoriť z testovania krátke hlásenie

Tieto úlohy odľahčujú činnosť vývojára od nezáživných stále sa opakujúcich úloh, ako aj od hľadania chýb v testovacom výstupe. V skutočnosti väčšina vývojových prostredí dnes vykonáva testovanie automaticky a bez vedomia vývojára, jedná sa však hlavne o testovanie programovacích praktík, ktoré nevyplývajú z implementovanej funkčnosti programu ale z definovaných štandardov a odporúčaní pre daný programovací jazyk. Osobne sa mi stalo že vývojové prostredie odhalilo programovú slučku ktorá sa v skutočnosti mohla vykonať iba raz, čo je často veľmi záludná chyba.

Aspektovo-orientovaný vývoj a testovanie

Popri skúmaní známych prvkov v testovaní chcem vyzdvihnúť nové možnosti v oblasti testovania ktoré ponúka aspektovo orientovaný vývoj softvéru. Klasické testovacie prostriedky a metódy sa snažia nezasahovať do testovaného programu, nakoľko vkladať testovacie (či iné redundantné) príkazy priamo do programu vedie k zahlteniu programu, dokonca môže spôsobiť neočakávané správanie programu.

Ako výborný príklad je dobré uviesť jednotkové testy, ktorých úlohou je definovať vstupy a k nim očakávané výstupy určitej jednotky a pomocou nich zhodnotiť jej funkčnosť. Tieto testy nijako nezasahujú do vnútra jednotky. Je možné ich kedykoľvek odstrániť či upraviť, bez zmeny funkcionality testovaného programu. V tomto prístupe je možné pozorovať dva nedostatky:

- Z výstupu testovania nie je možné zistiť cestu programu ktorá viedla k chybe, hlavne ak sa chyba prejaví neskôr než vznikne
- Z výstupu testovania nie je možné získať informácie o vnútornom stave jednotky počas testovania. Vidíme iba vstupy a výstupy.

Jednotkové testy sa dajú označiť za testovanie „čiernou skrinkou“, nakoľko nevidíme do vnútra testovanej jednotky. Vývojár však môže využiť svoje znalosti vnútra jednotky a vytvoriť tak testovanie „sivou skrinkou“. Riešením týchto problémov je práve aspektovo-orientovaný vývoj. Pri tejto forme vývoja je možné systematickým prístupom zasahovať do činnosti programu. Zároveň je možné rozdeliť jednotku programu na aspekty ktoré definujú túto jednotku. Testovanie takto priamo zasahuje do testovaného programu, ale toto zasahovanie sa vykonáva systematickým presne definovaným prístupom. Testovanie sa tak stáva integrálnou súčasťou vývoja, rovnako ako iné aspekty cieľového programu ako bezpečnosť či škálovateľnosť. Aspektový vývoj má rovnaké

výhody v oblasti testovania ako jednotkové testy, ale stiera nedostatky jednotkového testovania. Umožňuje oddeliť testovanie od testovaného programu, ale nebráni nám cielene zasiahnuť do testovaného programu za účelom získania jeho vnútorného stavu a správania. Aspektovo orientovaný prístup mení testovanie na testovanie pomocou bielej skrinky, nakoľko môžeme sledovať a meniť vnútorný stav testovaného procesu.

Aspektový prístup prináša ešte ďalšiu významnú zmenu v oblasti testovania a to možnosť rozdeliť testovaný modul na jeho aspekty. Nemusíme testovať modul ako celok, ale zameriame sa len na jeho dôležité aspekty, teda na základnú funkčnosť, či bezpečnosť. Potrebné testovanie je vždy možné jednoducho rozšíriť. Ak sa bude vykonávať takéto testovanie je dobré myslieť na aspekty už pri analýze a návrhu riešenia.

Ak sa pozrieme do publikácií o aspektovo-orientovanom vývoji, takmer každá z nich začína vysvetľovať tento prístup práve na použití monitorovania či overovania behu programu, pozri [7]. Preto dúfam že v budúcnosti sa bude klásť čoraz väčšia dôležitosť použitia tejto techniky pri vývoji softvéru.

Použitie tohto prístupu v malom tíme je plne závislé od snahy členov naučiť sa tento prístup, nakoľko ešte nie je všeobecne používaný a mnohí vývojári sa mu vyhýbajú. V spojení s agilnými metódami vývoja, akou je aj metóda SCRUM vytvára tento prístup synergický efekt v oblasti refaktorizácie – čiže riadenej zmeny existujúceho programu. Odporúčam ho zaviesť iba v prípade že každý člen malého tímu má základné znalosti z tejto problematiky.

Záver

Je potrebné testovanie pre vytvorenie funkčného a správneho programu? Nie, avšak pri testovaní je pravdepodobnosť tohto cieľa oveľa vyššia, pretože overenie správnosti je možné dosiahnuť iba testovaním. Avšak testovanie nikdy neoverí správnosť programu vo všetkých situáciách. Navyše s vývojom softvérových technológií stúpa komplexnosť riešení a tým aj potreba a náročnosť overovania testami.

Ako sme v tejto publikácii zistili, nedostatky v testovaní môžu vzniknúť nepochopením spôsobu testovania vývojárom, alebo zanedbaním niektorej časti prístupu k testovaniu. Preto je potrebné definovať prístupy testovania ako aj použité prostriedky pred začatím projektu. To môže spôsobiť edukačný charakter začiatku projektu, a slúži aj na zjednotenie názorov v tíme.

Existujú vývojové prístupy ktoré úspešne využívajú testovanie? Áno, jedná sa napríklad o metódu SCRUM pri ktorej je testovanie neustálou súčasťou vývoja. Pri tejto metóde sa vyžaduje sto percentné ukončenie úlohy pred prechodom na ďalšiu úlohu, pričom ukončenie úlohy je potrebné definovať pomocou testov. Vývoj je porovnateľný s testami riadeným vývojom, avšak nie je to povinnosť.

Stačia nám súčasné technológie k testovaniu? Dospeli sme k poznatku že jednotkové testy ktoré sú používané pri testami riadenom vývoji a aj iných testoch nedokážu pokryť testovanie vnútorného správania programu v dostatočnej miere, nakoľko sledujú len vstupné a výstupné charakteristiky programu. Preto navrhujem testovanie pomocou aspektovo-orientovaného prístupu, ktorý rozširuje možnosti jednotkových testov a dokáže zasahovať do testovaného programu veľmi systematickým prístupom.

Použitá literatúra

1. E.F. Miller, „Introduction to Software Testing Technology“, Tutorial: „Software Testing & Validation Techniques“, Druhé vydanie, IEEE EHO 180-0, 4-16
2. G.J. Myers, T. Badgett, T.M. Thomas, C. Sandler, „The Art of Software Testing“, Druhé vydanie, John Wiley and Sons Inc. , 2004
3. J. Tuomikoski, I. Tervonen, „Absorbing Software Testing into the Scrum Method“, University of Oulu, 2009
4. L. Williams, „Agile/Automated Testing“, NC State University, 2004
5. L. Luo, „Software Testing Techniques“,Technology maturation and Research Strategies, Carniege Mellon University Pittsturg, 2001
6. M. Lacey „How do we know we are done?“, 2008 , internet: http://www.scrumalliance.org/articles/107-how-do-we-know-when-we-are-done
7. R. Laddad, „AspectJ in action“, kapitola 5, Manning, 2003

Annotation

Testing – Hardest task in software development

In this essay we explore advantages and disadvantages of software testing, while we look at complications and simplifications of testing in actual software development. We search for solutions of some basic testing problems. We look at possibilities of testing in small team working on a short-time project. We try to find methodologies of testing that ease resting in such team. We describe the SCRUM methodology of software development, which abstractly defines used testing approaches. Advantages and disadvantages of test driven development are evaluated, because they are mostly used in this methodology. Before end we try to offer an alternative testing approach, which uses aspect oriented approach of software development and evaluate positive and negative effects.